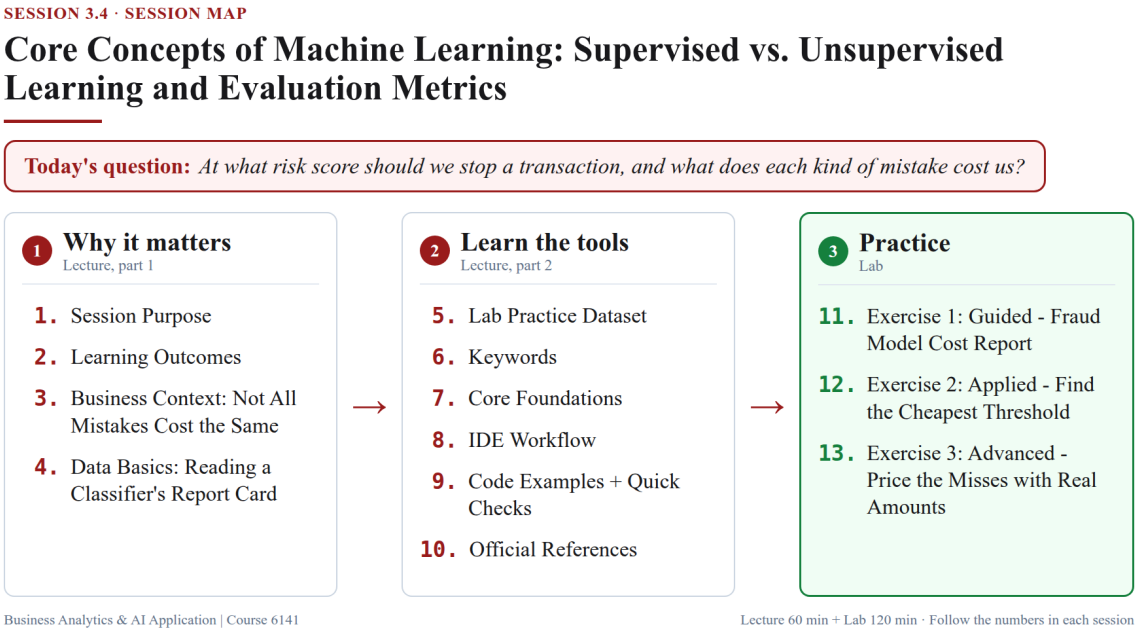
Session 3.4: Core Concepts of Machine Learning: Supervised vs. Unsupervised Learning and Evaluation Metrics

Course 6141: Business Analytics & AI Application | Reading for Week 12



Schedule: Week 12 | In-person classes (1:2 Delivery Ratio: 60 Min Lecture + 120 Min Lab)

1. Session Purpose

- A card issuer's fraud model gives every transaction a risk score from 1 to 100; above a cut-off score (the threshold), the transaction is held and the customer gets a verification text
- The two possible mistakes have very different costs: a false alarm costs about $5 of checking, while a missed fraud costs the stolen money
- The fraud operations manager asks: at what risk score should we stop a transaction, and what does each kind of mistake cost us? Today, you judge a model by business cost, not by a single accuracy number

2. Expected Learning Outcomes (By the end of this session, you will be able to)

1. "Which kind of learning fits this problem?": tell supervised learning (learning from examples with known answers) from unsupervised learning (finding groups without answers), and group transactions with K-Means
2. "How good is the risk score, really?": read a confusion matrix (a 2x2 count of right and wrong decisions) and calculate precision, recall, and ROC-AUC
3. "Where should we set the threshold?": choose the threshold with the lowest business cost, and fix the `ValueError` raised when scores are passed in place of 0/1 decisions

3. Business Context: Not All Mistakes Cost the Same

- Every decision on a transaction lands in one of four boxes:

	Model says: fraud (hold it)	Model says: legitimate (let it pass)
Really fraud	TP (True Positive): fraud caught	FN (False Negative): fraud missed, the money is lost
Really legitimate	FP (False Positive): false alarm, about $5 to check	TN (True Negative): normal purchase, no cost

- In our cost model, a missed fraud costs `$1,500` and a false alarm costs `$5`, so one miss costs as much as 300 false alarms.
- The accuracy trap: if only 1% of transactions are fraud, a model that says "never fraud" is 99% accurate and catches nothing. Accuracy hides the costly mistakes.

4. Data Basics: Reading a Classifier's Report Card

- Four measures, each answering one question:

Measure	Formula	Question it answers
Accuracy	(TP + TN) / all	What share of all decisions was right?
Precision	TP / (TP + FP)	Of the transactions we held, what share was really fraud?
Recall	TP / (TP + FN)	Of all the fraud, what share did we catch?
ROC-AUC	area under the ROC curve, from 0.5 to 1.0	How well does the score rank fraud above legitimate transactions, at every threshold?

- A higher threshold holds fewer transactions: fewer false alarms (higher precision) but more missed fraud (lower recall). The threshold moves this trade-off; the business cost decides where it should sit.

- ROC-AUC (Area Under the Receiver Operating Characteristic Curve): `0.5` means the score ranks no better than a coin flip; `1.0` means every fraud scores above every legitimate transaction.
- Supervised vs unsupervised:
  - **Supervised:** past examples come with the answer (`Actual_Fraud_Binary`), and the model learns to predict it. Today's risk score is a supervised model's output.
  - **Unsupervised:** no answer column; the method finds groups of similar rows by itself. K-Means is the most common example.

5. Lab Practice Dataset

- File Link: `data/session_03_04_dataset.csv`
- Audited 16-row card transaction dataset (fictional teaching data), one row per transaction, checked afterwards by fraud investigators:
  - `Transaction_ID`: Transaction code (e.g., `F-TXN-8001`).
  - `Transaction_Amount_USD`: Purchase amount in U.S. Dollars.
  - `Foreign_IP_Flag`: `1` if the purchase came from a foreign IP (Internet Protocol) address, the network address of the device; `0` if domestic.
  - `Failed_Login_Attempts_Count`: Failed login attempts before the purchase.
  - `Cardholder_Risk_Score`: The fraud model's risk score, from 1 (safe) to 100 (very risky).
  - `Actual_Fraud_Binary`: The investigators' answer: `1` = fraud, `0` = legitimate. 7 of the 16 transactions were fraud.

6. Keywords (Core Programming Concepts)

- `Supervised vs Unsupervised Learning`:
  - What: learning with known answers vs finding groups without answers
  - Why: the first question to ask about any ML project
- `Threshold`:
  - What: the cut-off score above which the model's answer becomes "yes"
  - Why: the business sets it, not the model
- `Confusion Matrix`:
  - What: the 2x2 count of TP, FP, FN, and TN
  - Why: shows each kind of mistake separately, so each can be priced
- `Precision` and `Recall`:
  - What: the share of holds that were fraud, and the share of fraud that was held
  - Why: the two sides of the trade-off a threshold controls
- `ROC-AUC`:
  - What: one number for how well a score ranks positives above negatives
  - Why: compares scoring models before any threshold is chosen

- `K-Means Clustering`:
  - What: an unsupervised method that splits rows into K groups of similar values
  - Why: finds customer or transaction segments when no labels exist

7. Core Programming & Computational Foundations

- **7.1 From a score to a decision: the threshold**
  - *The idea:* A fraud model does not say "fraud" or "not fraud". It gives each transaction a score, here from 1 to 100. The business then chooses a **threshold**: every transaction scoring at or above it is held for checking. The model ranks; the business decides where to draw the line.
  - *See it on real data:* the 9 highest-scoring transactions, sorted by score.

Transaction	Score	Really fraud?	Held at 70?	Held at 75?
`F-TXN-8006`	`22`	no	no	no
`F-TXN-8009`	`71`	no	**yes** (false alarm)	no
`F-TXN-8005`	`74`	yes	yes	**no** (missed fraud)
`F-TXN-8015`	`76`	yes	yes	yes
`F-TXN-8001`	`78`	yes	yes	yes
`F-TXN-8011` to `F-TXN-8007`	`85` to `92`	yes	yes	yes

    Moving the line from 70 to 75 removes one false alarm but lets one fraud through.
  - *In code:* `(df["Cardholder_Risk_Score"] >= 70).astype(int)` turns every score into `1` (hold) or `0` (pass).
  - *More examples:*

Threshold	Transactions held	Who they are
`50`	`8`	all 7 frauds + 1 false alarm
`70`	`8`	the same 8
`72`	`7`	exactly the 7 frauds
`80`	`4`	3 frauds slip through
`90`	`1`	only the riskiest one

  - *Remember:* the model gives scores; the threshold turns them into decisions.
- **7.2 The confusion matrix: four kinds of outcome**
  - *The idea:* Every decision lands in one of four boxes. Counting them separately shows not only how often the model is right, but **which** mistakes it makes, and those mistakes have different costs.

	Held (`1`)	Passed (`0`)
Really fraud	TP: fraud caught	FN: fraud missed
Really legitimate	FP: false alarm	TN: normal purchase passed

  - *See it on real data:* at threshold 70, `confusion_matrix(y_true, y_pred)` returns `[[8 1] [0 7]]`. Scikit-Learn puts the "really legitimate" row first:

	Passed	Held
Really legitimate	TN `8`	FP `1`
Really fraud	FN `0`	TP `7`

  - *Common mistake:* reading the matrix in the wrong order. `.ravel()` flattens it as `tn, fp, fn, tp`, always in that order.
  - *More examples:* the four counts at other thresholds.

Threshold	TP	FP	FN	TN
`50`	`7`	`1`	`0`	`8`
`72`	`7`	`0`	`0`	`9`
`80`	`4`	`0`	`3`	`9`
`90`	`1`	`0`	`6`	`9`

  - *Remember:* TP and TN are right; FP and FN are the two different mistakes.

- **7.3 Accuracy, precision, and recall**
  - *The idea:* Three measures answer three different questions:
    - **Accuracy** = (TP + TN) / all: what share of all decisions was right?
    - **Precision** = TP / (TP + FP): of the transactions we held, what share was really fraud?
    - **Recall** = TP / (TP + FN): of all the fraud, what share did we catch?
  - *See it on real data:*

Threshold	Accuracy	Precision	Recall
`70`	`0.938`	`0.875`	`1.000`
`75`	`0.938`	`1.000`	`0.857`
`80`	`0.812`	`1.000`	`0.571`
`90`	`0.625`	`1.000`	`0.143`

    Raising the threshold makes every hold more certain (precision up) but lets more fraud through (recall down).
  - *The accuracy trap:* a "model" that never holds anything is right on every legitimate transaction: `9` of `16`, an accuracy of `0.562`, while catching no fraud at all. In real card data, where fraud is often under 1% of transactions, such a useless model would be over 99% accurate.
  - *More examples:* the F1-score combines precision and recall into one number (their harmonic mean); it is high only when both are high.

Threshold	F1-score
`50`	`0.933`
`72`	`1.000`

Threshold	F1-score
75	0.923
80	0.727
90	0.250

  - *Remember:* precision asks "were the holds right?"; recall asks "did we catch the fraud?"

- **7.4 ROC-AUC: ranking quality at every threshold**
  - *The idea:* ROC-AUC (Area Under the Receiver Operating Characteristic Curve) judges a score before any threshold is chosen. It answers: if you pick one fraud and one legitimate transaction at random, how often does the fraud get the higher score? `0.5` is a coin flip; `1.0` means every fraud scores above every legitimate transaction.
  - *See it on real data:* `roc_auc_score(y_true, scores)` takes the scores themselves, not 0/1 decisions.

Score used	ROC-AUC	Meaning
`Cardholder_Risk_Score`	`1.000`	every fraud (`74-92`) above every legitimate one (`12-71`)
`Transaction_Amount_USD`	`0.984`	amount alone ranks almost as well
`Failed_Login_Attempts_Count`	`0.984`	so do failed logins
`Foreign_IP_Flag`	`0.944`	a yes/no flag ranks less finely

  - *Common mistake:* passing 0/1 decisions to `roc_auc_score()`, or scores to `confusion_matrix()`. The first loses information; the second raises `ValueError: Classification metrics can't handle a mix of binary and continuous targets`.
  - *More examples:*

Code	Result	What it tells you
`roc_auc_score(y_true, 100 - scores)`	`0.0`	a score that is perfectly backwards
`roc_auc_score(y_true, [50] * 16)`	`0.5`	the same score for everyone: a coin flip
`roc_auc_score(y_true, scores / 100)`	`1.0`	rescaling does not change the ranking

  - *Remember:* AUC compares scores; precision and recall judge decisions.

- **7.5 Unsupervised learning: groups without answers**
  - *The idea:* Supervised learning learns from examples **with** answers, such as investigated fraud cases. Unsupervised learning has no answer column: it looks for groups of similar rows by itself. K-Means, the most common method, places K group centers and assigns every row to its nearest center, moving the centers until the groups stop changing.

– *See it on real data:* K-Means on amount and risk score only (the fraud column is not used).

K (number of groups)	Group sizes	Inertia (spread inside the groups; lower is tighter)
2	8, 8	3.93
3	4, 8, 4	0.70
4	4, 8, 2, 2	0.38

Inertia always falls as K grows, but the big drop is from 2 to 3 groups; after that, extra groups only split small groups further. This "elbow" suggests K = 3.

– *Common mistake:* expecting K-Means to name the groups. It only numbers them `0`, `1`, `2`; a person reads each group's profile and gives it a business name, such as "small, safe purchases".

– *More examples:* `kmeans.predict()` places new rows (scaled the same way) into the existing groups.

Code or input	Result	What it tells you
a new purchase: `$5,000`, score `95`	group `2`	large, high-risk purchases
a new purchase: `$40`, score `10`	group `1`	small, safe purchases
a new purchase: `$1,900`, score `73`	group `0`	mid-size, higher-risk purchases
`kmeans.cluster_centers_.shape`	`(3, 2)`	3 group centers, 2 features each

– *Remember:* supervised = learn the answer; unsupervised = find the groups, then name them.

8. Development Workflow in Google Antigravity IDE

- `[1]` File Explorer: create your script with your name and the session code (for example, `Reacher_S34_demo_1.py`), and check the dataset is in `data/`.
- `[2]` Code Editor: write a few lines, run, repeat. Small steps keep errors small.
- `[3]` AI Agent Chat: on an error, capture the traceback (Python's error report) with `Win + Shift + S` (macOS: `Cmd + Shift + 4`), paste it with `Ctrl + V`, and ask why it failed.
- `[4]` Terminal: run `python` followed by your script name. Return code `0` means the script finished with no errors. First time only: `pip install scikit-learn`.

9. Hands-On Coding Micro-Examples

- Every example has three parts: **1. Input**, **2. Process**, and **3. Output**. Name each file with your own name and the session code, for example `Reacher_S34_demo_1.py` (pattern: `YourName_S34_demo_1.py`).
- *Micro-Example 1 (Confusion Matrix at Threshold 70):* `YourName_S34_demo_1.py`

```python
# 1. Input: the investigators' answers and the model's scores
import pandas as pd
from sklearn.metrics import confusion_matrix, precision_score, recall_score
df = pd.read_csv("data/session_03_04_dataset.csv")
y_true = df["Actual_Fraud_Binary"]

# 2. Process: hold every transaction scoring 70 or more
y_pred = (df["Cardholder_Risk_Score"] >= 70).astype(int)
cm = confusion_matrix(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)

# 3. Output: the 2x2 table, then the two key measures
print("Rows: really legitimate, really fraud | Columns: passed, held\n", cm)
print(f"Precision: {precision:.3f} | Recall: {recall:.3f}")
```

  - *How it works, line by line:*
    - `y_true` holds the investigators' answers: `1` for fraud, `0` for legitimate.
    - `(df["Cardholder_Risk_Score"] >= 70).astype(int)` turns each score into a decision at threshold 70 (section 7.1).
    - `confusion_matrix(y_true, y_pred)` counts the four outcomes (section 7.2).
    - `precision_score()` and `recall_score()` compute the two key measures (section 7.3).
  - *Read the output:*

```
Rows: really legitimate, really fraud | Columns: passed, held
 [[8 1]
 [0 7]]
Precision: 0.875 | Recall: 1.000
```

    - Top row: 8 legitimate transactions passed, 1 held by mistake (`F-TXN-8009`, score `71`).
    - Bottom row: 0 frauds missed, all 7 held.
    - Recall `1.000`: every fraud was caught. Precision `0.875`: 7 of the 8 holds were real fraud.

- *Quick Check 1 (`classification_report()`):*
  - New tool: `classification_report(y_true, y_pred)` prints precision, recall, and the F1-score (one number that balances precision and recall) for both classes, plus accuracy. Why: the full report card in one call.
  - Task: add `classification_report` to the import line, then add one line to part **3. Output**: `print(_____(y_true, y_pred))`
  - Check: the `accuracy` line shows `0.94`, and the row for class `1` shows precision `0.88` and recall `1.00`.

- *Micro-Example 2 (ROC-AUC: Ranking Quality at Every Threshold):* `YourName_S34_demo_2.py`

```python
# 1. Input: the answers and the scores
import pandas as pd
from sklearn.metrics import roc_auc_score
```

```
df = pd.read_csv("data/session_03_04_dataset.csv")
y_true = df["Actual_Fraud_Binary"]

# 2. Process: AUC uses the scores directly, not 0/1 decisions
auc = roc_auc_score(y_true, df["Cardholder_Risk_Score"])

# 3. Output: compare the score ranges of the two groups
print(f"ROC-AUC: {auc:.3f}")
print(df.groupby("Actual_Fraud_Binary")["Cardholder_Risk_Score"].agg(["min",
↪  "max"]))
```

  - *How it works, line by line:*
    - `roc_auc_score(y_true, df["Cardholder_Risk_Score"])` uses the raw scores, because it checks every possible threshold at once (section 7.4).
    - `df.groupby("Actual_Fraud_Binary")["Cardholder_Risk_Score"].agg(["min", "max"])` shows the lowest and highest score in each group, to see why the AUC is what it is.
  - *Read the output:*

```
ROC-AUC: 1.000
                      min  max
Actual_Fraud_Binary
0                      12   71
1                      74   92
```

    - Legitimate transactions score from `12` to `71`; fraud from `74` to `92`. The two ranges do not overlap, so the ranking is perfect: AUC `1.000`.
    - Any threshold from `72` to `74` would separate them exactly. With only 16 transactions, treat this as encouraging, not as proof.

- *Quick Check 2 (`accuracy_score()`):*
  - New tool: `accuracy_score(y_true, y_pred)` returns the share of correct decisions. Why: shows how a single accuracy number can look good while hiding the costly mistakes.
  - Task: add two lines to part **3. Output**: `from sklearn.metrics import accuracy_score` and `print(_____(y_true, (df["Cardholder_Risk_Score"] >= 90).astype(int)))`
  - Check: the Terminal shows `0.625`. At threshold 90, 62.5% of the decisions are right, yet 6 of the 7 frauds pass through.

- *Micro-Example 3 (K-Means: Groups Without Answers):* `YourName_S34_demo_3.py`

```
# 1. Input: two features only; the fraud column is NOT used
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
df = pd.read_csv("data/session_03_04_dataset.csv")
features = df[["Transaction_Amount_USD", "Cardholder_Risk_Score"]]
```

```
# 2. Process: scale, then form 3 groups of similar transactions
scaled = StandardScaler().fit_transform(features)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df["Cluster_ID"] = kmeans.fit_predict(scaled)

# 3. Output: the average profile of each group, and its fraud share
profile = df.groupby("Cluster_ID")[
    ["Transaction_Amount_USD", "Cardholder_Risk_Score", "Actual_Fraud_Binary"]
].mean().round(1)
print(profile)
```

  - *How it works, line by line:*
    - Only two features go in: amount and risk score. The fraud column is deliberately left out; this is unsupervised learning (section 7.5).
    - `StandardScaler().fit_transform(features)` puts both on one scale (Session 3.2), so the dollar amounts do not outweigh the scores.
    - `KMeans(n_clusters=3, random_state=42, n_init=10)` asks for 3 groups; `n_init=10` tries 10 starting points and keeps the best; `random_state=42` makes the result repeatable.
    - `.fit_predict(scaled)` finds the groups and returns each row's group number, stored as `Cluster_ID`.
    - The `groupby()` table averages each group's columns, including the fraud column, which K-Means never saw.
  - *Read the output:*

```
            Transaction_Amount_USD  Cardholder_Risk_Score
 ↪  Actual_Fraud_Binary
Cluster_ID
0                           1795.0                   74.8
 ↪  0.8
1                             52.2                   16.1
 ↪  0.0
2                           3587.5                   88.5
 ↪  1.0
```

    - Group `1`: small, low-risk purchases, no fraud. Group `0`: mid-size, higher-risk purchases, 80% fraud. Group `2`: large, high-risk purchases, all fraud.
    - The group numbers have no meaning in themselves; a person names the groups from these profiles.
    - Without seeing the answers, K-Means found groups that line up with fraud: a sign that amount and score carry the fraud pattern.

- *Quick Check 3 (`.size()`):*
  - New tool: `df.groupby("Column").size()` counts the rows in each group. Why: a segment's size matters as much as its profile.
  - Task: add one line to part **3. Output**: `print(df.groupby("Cluster_ID").____())`
  - Check: group sizes `4`, `8`, and `4`.

10. Official Python Documentation & References

- Scikit-Learn: Classification Metrics (Confusion Matrix, Precision-Recall).
- Scikit-Learn: Clustering (K-Means Algorithm Overview).
- Fawcett: An Introduction to ROC Analysis (Pattern Recognition Letters).

11. Exercise 1: Guided Practice - Fraud Model Cost Report

- *Step 1: Open Script:* In Google Antigravity IDE, create a new script named `YourName_S34_evaluation.py`, replacing `YourName` with your own name (for example, `Reacher_S34_evaluation.py`).
- *Step 2: Count the Outcomes and Price the Mistakes:* Type the following code:

```
# 1. Input: the answers and the scores
import pandas as pd
from sklearn.metrics import confusion_matrix, precision_score, recall_score
df = pd.read_csv("data/session_03_04_dataset.csv")
y_true = df["Actual_Fraud_Binary"]

# 2. Process: decisions at threshold 70, then the cost of the mistakes
y_pred = (df["Cardholder_Risk_Score"] >= 70).astype(int)
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
false_alarm_cost = 5.0
missed_fraud_cost = 1500.0
total_cost = fp * false_alarm_cost + fn * missed_fraud_cost

# 3. Output: counts, measures, and the cost
print(f"TP={tp}, FP={fp}, FN={fn}, TN={tn}")
print(f"Precision={precision_score(y_true, y_pred):.2f}, "
      f"Recall={recall_score(y_true, y_pred):.2f}")
print(f"Cost of mistakes: ${total_cost:,.2f}")
```

  - Two strings written next to each other inside `print()` are joined into one; this keeps long lines readable.

- *Step 3: Interpret:* In two comment lines, explain why one missed fraud costs as much as 300 false alarms, and what that means for where the threshold should sit.
- *Step 4: Execute in Terminal:* Run `python YourName_S34_evaluation.py` in Panel `[4]` and confirm return code `0`.
- *Success Check:* `TP=7, FP=1, FN=0, TN=8`; `Precision=0.88, Recall=1.00`; `Cost of mistakes: $5.00`.

12. Exercise 2: Applied Practice - Find the Cheapest Threshold

- *Coding Challenge:* In `YourName_S34_practice.py`, test thresholds from 50 to 90 in steps of 5 and find the one with the lowest cost.
- *Task 1 (Threshold Loop):* Loop with `for threshold in range(50, 95, 5):`. For each threshold, compute the decisions, `tn, fp, fn, tp`, and the cost (false alarm `$5`, missed fraud `$1,500`). Append one dictionary per threshold to a list, and turn the list into a DataFrame with `Threshold`, `TP`, `FP`, `FN`, `TN`, and `Cost_USD`. `range(50, 95, 5)` gives 50, 55, …, 90; the stop value 95 itself is not included.
- *Task 2 (Read the ValueError & AI Diagnosis):* Inside the loop, try `confusion_matrix(y_true, df["Cardholder_Risk_Score"] / 100)`, passing the scores instead of the decisions. The script stops with `ValueError: Classification metrics can't handle a mix of binary and continuous targets`. Capture the traceback with `Win + Shift + S`, paste it into Panel `[3]` (AI Agent Chat) with `Ctrl + V`, and ask how to turn scores into decisions.
- *Task 3 (Choose and Save):* Fix the call with `(score >= threshold).astype(int)`. Choose the threshold with the lowest cost; if several tie, choose the highest of them, because it sends the fewest customers to verification on future transactions. Save the table to `data/fraud_threshold_costs.csv` with `index=False`.
- *Success Check:* thresholds 50 to 70 cost `$5` (1 false alarm); 75 costs `$1,500`; 80 and 85 cost `$4,500`; 90 costs `$9,000`. Chosen threshold: `70`.

13. Exercise 3: Advanced Practice - Price the Misses with Real Amounts

- *Goal:* The flat `$1,500` per missed fraud is a guess. Replace it with each missed transaction's real amount, test every possible threshold, and show the manager how much the guess was off.
- *What you need:*
  - File: `data/session_03_04_dataset.csv` (described in **5. Lab Practice Dataset**).
  - Script name: `YourName_S34_challenge.py`.
  - Tools you already know: Boolean masks with `&` and `~` (Session 1.4), `.sum()`, `for` loops, `range()`, lists of dictionaries, and `pd.DataFrame()`.
- *Requirements (follow them in order):*
  1. For every whole-number threshold from 1 to 100, mark the held transactions: `held = df["Cardholder_Risk_Score"] >= threshold`.
  2. Compute two costs for each threshold:
     - `Flat_Cost_USD`: false alarms x `$5` + missed frauds x `$1,500`
     - `Real_Cost_USD`: false alarms x `$5` + the **sum of `Transaction_Amount_USD`** of the missed frauds
  3. Put all 100 results in one DataFrame with `Threshold`, `False_Alarms`, `Missed_Frauds`, `Flat_Cost_USD`, and `Real_Cost_USD`.
  4. Print every threshold with the lowest `Real_Cost_USD`.
  5. Print the rows for thresholds `70`, `75`, `80`, and `90`, and add `Gap_USD` = `Real_Cost_USD - Flat_Cost_USD`.
  6. In two comment lines, recommend a threshold and explain why the flat guess is risky.
- *New tools for this exercise:*

  - A false alarm is a held legitimate transaction: `held & (df["Actual_Fraud_Binary"] == 0)`. A missed fraud is a passed fraud: `~held & (df["Actual_Fraud_Binary"] == 1)`. `~` flips `True` and `False`.
  - Summing money over a mask: `df.loc[missed, "Transaction_Amount_USD"].sum()` adds up the amounts of the missed frauds only.
  - Selecting several rows by value: `results[results["Threshold"].isin([70, 75, 80, 90])]`.

- *Hints:*
  - Inside the loop, compute the two masks first, then both costs, then append one dictionary.
  - `range(1, 101)` gives 1, 2, …, 100.

- *Self-Check (expected results):*
  - Lowest real cost: `$0` at thresholds `72`, `73`, and `74`: every fraud held and no false alarm.
  - Gaps: threshold 70: `$0` (`$5` both ways); 75: `$600` (`$1,500` flat vs `$2,100` real); 80: `$830` (`$4,500` vs `$5,330`); 90: `$6,480` (`$9,000` vs `$15,480`).
  - Business meaning: the missed frauds at high thresholds are the large ones, so the flat guess understates the loss more and more as the threshold rises. On these 16 past transactions the safest choice is `72` to `74`, but a perfect result on so few rows will not last; re-check the threshold every month on new investigated cases, and keep using real amounts in the cost.